Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА
Институт Информационных Технологий
Кафедра Промышленной Информатики

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ

Тема лекции «Основы программирования на языке C++»

Лектор Каширская Елизавета Натановна (к.т.н., доцент, ФГБОУ ВО "МИРЭА - Российский технологический университет") e-mail: liza.kashirskaya@gmail.com

Лекция № 2

Чем отличается C от C++?

На 99% — ничем, но в C++ есть родная поддержка объектно-ориентированного программирования (ООП).

C++ — это улучшенный C. У этих языков одинаковый на 99% синтаксис и команды, но C — это больше про структурное и процедурное программирование, а C++ — про объектно-ориентированное.

C — язык, который сделал в 1973 году Деннис Ритчи. Главная цель языка — скорость, быстродействие и универсальность. Язык изначально проектировался как системный, чтобы на нём можно было писать код для процессоров, драйверов и создавать на нём операционные системы. В то время большинство этих вещей делали на ассемблере, и Ритчи хотел это упростить.

C++ придумал Бьёрн Страуструп в начале восьмидесятых, когда ему не хватало возможностей стандартного C. Он сделал язык более строгим, добавил в него классы, ООП-подход и перегрузку операторов, сохранив скорость оригинального C. В 1983 году Бьёрн переименовал язык из «C с классами» в C++.

Программа – это реализация алгоритма для выполнения задачи компьютером (ЭВМ).

С помощью программы мы формулируем алгоритм на языке, понятном компьютеру. Таким языком служит язык программирования.

На сегодняшний день одним из самых распространенных и востребованных языков программирования является язык С++.

На языке С++ можно составлять программы для инженерных расчетов, также можно строить оконные проекты, имеющие пользовательский графический интерфейс.

Этапы решения задачи на ЭВМ

В процессе создания любой программы можно выделить несколько этапов.

1) Постановка задачи. Этот этап выполняется специалистом в предметной области на естественном языке. Необходимо определить цель задачи, ее содержание и общий подход к решению. Возможно, что задача решается точно (аналитически) и без компьютера. Уже на этапе постановки надо учитывать эффективность алгоритма решения задачи на ЭВМ, ограничения, накладываемые аппаратным и программным обеспечением.

2) Анализ задачи и моделирование. На этом этапе определяются исходные данные и ожидаемый результат, выявляются ограничения на их значения, выполняется формализованное описание задачи и построение или выбор математической модели, пригодной для решения на компьютере.

3) Разработка или выбор алгоритма решения задачи выполняется на основе ее математического описания. Многие задачи можно решить различными способами. Программист должен выбрать оптимальное решение. Неточности в поставке, анализе задачи или разработке алгоритма могут привести к скрытой ошибке: программист получит неверный результат, считая его правильным.

4) Проектирование общей структуры программы. На этом этапе формируется модель решения, с последующей детализацией и разбивкой на подпрограммы, определяется архитектура программы, способ хранения информации (набор переменных, констант, массивов данных и так далее).

5) Кодирование представляет собой запись алгоритма на языке программирования.

Современные системы программирования позволяют ускорить процесс разработки программы, автоматически создавая часть ее текста. Однако творческая работа лежит на программисте. Для успешной реализации цели проекта программисту необходимо использовать методы структурного программирования, то есть программирования с использованием подпрограмм, которые в языке программирования C++ носят названия функций.

6) Отладка и тестирование программы. Под откладкой понимается устранение ошибок в программе. Тестирование позволяет вести их поиск и в конечном счете убедиться в том, что полностью отлаженная программа дает правильный результат. Для этого разрабатываются системы тестов - специально подобранных контрольных примеров с такими подборками параметров, для которых решение задачи известно. Тестирование должно охватывать все возможные ветвления в программе, то есть проверяют все ее инструкции и включают даже такие исходные данные, для которых решение невозможно. Проверка особых исключительных ситуаций необходима для анализа корректности работы программы. Например, программа должна отказать клиенту банка в просьбе выдать сумму, отсутствующую на его счете. В ответственных проектах большое внимание уделяется так называемой «защите от неумелого пользователя», подразумевающей устойчивость программы по отношению к неумелому обращению пользователя. Использование специальных программ-отладчиков, которые позволяют выполнять программу по отдельным шагам, просматривая при этом значения переменных, значительно упрощает этот этап.

7) Анализ результатов. Если программа выполняет моделирование какого-либо известного процесса, следует сопоставить результат вычислений с результатами наблюдений. В случае существенного расхождения необходимо изменить модель.

8) Публикация результатов работы, передача заказчику для эксплуатации.

9) Сопровождение программы. Этот конечный этап включает в себя консультации представителей заказчика по работе с программой и обучению персоналом. Недостатки и ошибки, выявленные в процессе эксплуатации, устраняются.

ТИПЫ ДАННЫХ

В языке C++ все переменные имеют определенный тип данных. Тип определяет множество значений, которые могут принимать объекты программы (константы и переменные), а также совокупность операций, допустимых над этими значениями. Например, значения 1 и 3 относится к целочисленному типу. Над данными целого типа можно выполнять любые арифметические операции.

А вот данные «отличное» или «учеба» принадлежат к строковому типу.

Тип данных присваивается переменной при ее объявлении или инициализации. Далее приведем основные типы данных языка C++, которые нам понадобятся.

Типы данных

В языке программирования C++ определены некоторые стандартные типы данных, которые представлены в табл. 7.

Таблица 7. Стандартные типы данных C++

Типы данных	Значения
int	целый тип, размер типа int не определяется стандартом, а зависит от компьютера и компилятора, для 16-разрядного процессора под величины этого типа отводится 2 байта, для 32-разрядного — 4 байта. Примеры значений типа int: 5, 0, -1, 100.
double	вещественный тип с двойной точностью. Типы данных с плавающей точкой хранятся в памяти компьютера иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей — мантиссы и порядка. Мантисса — это число, большее 1.0, но меньшее 10. Для величин типа double, занимающих 8 байт, под порядок и мантиссу отводится 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а длина порядка — его диапазон. Примеры значений типа double: 5.0, -0.00001, 2.9987.
float	вещественный тип. В компьютерах величины типа float занимают 4 байта, из которых один двоичный разряд отводится под знак мантиссы, 8 разрядов под порядок и 23 под мантиссу. Тип float имеет меньшую точность, чем double. В большинстве случаев лучше использовать double.
char	символьный тип, под величину символьного типа отводится количество байт, достаточное для размещения любого символа из набора символов для данного компьютера, что и обусловило название типа. Как правило, это 1 байт. Каждый символ имеет свой собственный целочисленный код, согласно таблице ASCII (англ. American Standard Code for Information Interchange).Примерызначенийтипа char: "A" (код 65), "7" (код 55), "-" (код 189), "/" (код 191).

ПЕРЕМЕННЫЕ В ПРОГРАММИРОВАНИИ

В программировании существует понятие «Переменная».

Переменная — это нечто такое, что способно изменять свое значение в ходе выполнения программы.

Возьмём самое обычные арифметическое выражение: $x = a + b$.

В этом выражении присутствуют три именованных неизвестных: x, a и b.

Каждое неизвестное в этом уравнении может изменять своё состояние: за названием неизвестного сокрыто какое-либо значение. Каким-то неизвестным значения задаём мы, какие-то неизвестные вычисляются по формулам с помощью ставшими изветными неизвестных. Любое изменение значения равносильно изменению состояния.

В компьютере почти все то же самое.

Переменная в языке C++ представляет собой название неизвестного, такого неизвестного, значение-состояние которому можно задавать или изменять во время работы программы.

ПЕРЕМЕННЫЕ В ПРОГРАММИРОВАНИИ

Информация в компьютере хранится в памяти. Минимальная единица информации хранится в одной ячейке памяти. У любой ячейки памяти есть свой собственный физический адрес. У каждой ячейки памяти свой индивидуальный физический адрес. Когда мы хотим, чтобы компьютер, например, выводил некоторое значение на монитор, нам необходимо заставить компьютер запомнить это значение.

Запоминаемое компьютером значение попадает в некоторую область памяти, где и хранится. Область памяти представляет собой целостный блок, собранный из какого-то числа ячеек. Хранимое внутри области памяти значение определяется компилятором с помощью первой ячейки.

У каждой ячейки есть собственный физический адрес. В C++ обращение к адресу первой ячейки некоторого блока памяти эквивалентно обращению ко всему блоку памяти.

Чтобы хранимое в блоке памяти значение, например, отображалось на экране, нужно вытащить значение из блока памяти. Чтобы вытащить значение из блока памяти, нужен адрес первой ячейки этого блока памяти. Обращением к адресу первой ячейки блока памяти всегда можно вытащить значение, хранимое внутри всего блока.

Для закрепления материала можно выполнить простое *упражнение*.

1. Объявите две переменные и присвойте им значения.

2. Поменяйте значения, хранимые в переменных.

Например, вы объявили a=5; b=10. Необходимо сделать так, чтобы при выводе значения b на монитор выводилось число 5, а при выводе значения a число 10.

ПЕРЕМЕННЫЕ В ПРОГРАММИРОВАНИИ

Зачем программисту нужны переменные? Чтобы программа могла что-то сделать, программе нужен доступ к какому-либо участку памяти компьютера. Например, чтобы выполнить простой расчет суммы из двух слагаемых, программе нужно место под хранение первого слагаемого, второго слагаемого и место под хранение полученного результата. То есть для вычисления суммы может требоваться либо два, либо три участка памяти (если используется два, то результат записывается поверх использованного слагаемого). Результат вычисления часто необходим для дальнейшего вычисления, в том числе для повторяемых вычислений.

Вот так выглядит простейшая простая программа вычисления суммы:

```
cout << 5+10 << "\n"; //Вывести на экран 15
```

Подход написания кодов подобным образом ведёт к тому, что нет возможности сохранять результаты для дальнейшего их использования. Вот получили мы в результате вычисления - число 15, а как этот результат использовать дальше? Числа 5 и 10 в этом примере являются константами: они где-то сохранены в памяти компьютера, иначе бы вычисление не представлялось возможным, но повторно обратиться к 5 и 10 мы уже не можем, потому что непосредственно с этими константами способа связи нет. Только считать заново!

Кроме этого, во время работы подобной программы вводить значения для вычислений с клавиатуры нельзя.

Поэтому, чтобы программы были полноценными, программисты почти всегда используют переменные. Редкая программа может оказаться полезной, если переменных в ней вообще нет.

Улучшение программы может быть таким:

```
int year_birthay = 1997;  //Переменная year_birthay инициализирована
// значением 1987
cout << year_birthay + 10;
```

Но сложно утверждать, что в таком коде много смысла, потому что используется значение, заданное программе в исходном коде. Это значение непосредственно во время работы программы не изменить никак. С тем же успехом можно было использовать, например, константу, поэтому этот пример мало отличается от предыдущего.

Тем не менее, этот пример ближе к полноценному коду. Ведь если требуется ввод с клавиатуры, то использование имени переменной позволяет связаться с участком памяти, в который компилятор сможет записывать изменения:

```
#include <iostream.h>
int main(){
  int year_birthay = 0;
  cout << "Введи год рождения: ";
  cin >> year_birthay; //Записали в память значение.
  cout << "Результат работы программы: " << year_birthay + 10 <<"\n";
//Использовали записанное значение
  return 0;
}
```

Благодаря возможности именования участков памяти мы можем написать почти любое название, которое будет удобно использовать в дальнейшем.

ПЕРЕМЕННЫЕ В ПРОГРАММИРОВАНИИ

При именовании переменных нужно иметь в виду следующее.

1. Первым символом имени переменной не может быть цифра.
2. Имя переменной может состоять только из символов латинского алфавита, цифр и символа подчёркивания.
3. Символы в верхнем и нижнем регистрах рассматриваются как разные.
4. В качестве имени нельзя использовать ключевые слова, зарезервированные компилятором.
5. Не стоит начинать название с одного или двух символов подчёркивания: подобные имена используются разработчиками для нужд компилятора, поэтому использования подобных имён способно привести к неожиданным последствиям
6. Стандарт не ограничивает длину задаваемого переменной имени, но некоторые платформы могут вводит свои ограничения на длину.

Объявление переменной в C++ происходит таким образом: сначала указывается тип данных для этой переменной, а затем название этой переменной.

Примеры объявления переменных

int a; // *объявление переменной a целого типа.*
float b; // *объявление переменной b типа данных с плавающей запятой.*

Инициализация переменной

При объявлении переменной мы можем присвоить ей значение в этот же момент. Это называется инициализацией переменной.

C++ поддерживает два основных способа инициализации переменных.

Способ № 1. Копирующая инициализация (или «инициализация копированием») с помощью знака равенства =:

```
int n = 5; // копирующая инициализация
double c = 14.2; // инициализация переменной типа double.
char d = 's'; // инициализация переменной типа char.
bool k = true; // инициализация логической переменной k.
```

Способ № 2. Прямая инициализация с помощью круглых скобок ():

```
int n(5); // прямая инициализация
```

Инициализация переменной

В C++ имеется важное отличие между понятиями определения (инициализации) и объявления переменной. Под объявлением понимается, собственно, задание имени переменной и указание ей нужного типа. Под определением понимается внесение какой-то информации в объявленную переменную.

Размеры переменных зависят от вашей операционной системы. Чтобы посмотреть размер переменной, существует оператор sizeof:

```
cout<<sizeof(int)<<"\n"; //Узнали размер переменной типа int
cout<<sizeof(double)<<"\n"; //Узнали размер переменной типа double
```

Объявление и инициализация нескольких переменных

В одном стейтменте можно объявить сразу несколько переменных одного и того же типа данных, разделяя их имена запятыми. *Например*, следующие 2 фрагмента кода выполняют одно и то же:

```
int a, b;
```

и

```
int a;
int b;
```

Кроме того, вы даже можете инициализировать несколько переменных в одной строке:

```
int a = 5, b = 6;
int c(7), d(8);
```

Объявление и инициализация нескольких переменных

Есть три ошибки, которые совершают новички при объявлении нескольких переменных в одном стейтменте.

Ошибка №1. Указание каждой переменной одного и того же типа данных при инициализации нескольких переменных в одном стейтменте. Это не критичная ошибка, так как компилятор легко её обнаружит и сообщит вам об этом.

```
int a, int b; // неправильно (ошибка компиляции)
int a, b; // правильно
```

Ошибка №2. Использование разных типов данных в одном стейтменте. Переменные разных типов должны быть объявлены в разных стейтментах. Эту ошибку компилятор также легко обнаружит:

```
int a, double b; // неправильно (ошибка компиляции)
int a; double b; // правильно (но не рекомендуется)
// Правильно и рекомендуется (+ «читабельнее»):
int a;
double b;
```

Ошибка №3. Инициализация двух переменных с помощью одной операции:

```
int a, b = 5; // неправильно (переменная a остаётся
// неинициализированной)
int a = 5, b = 5; // правильно
```

Заметьте, что в C++ оператор присваивания (=) не является знаком равенства и не может использоваться для сравнения значений. Оператор равенства записывается как «двойное равно: = =.

Присваивание используется для сохранения определенного значение в переменной. *Например*, запись вида *a* = 10 задает переменной a значение числа 10.

Где объявлять переменные? Старые версии компиляторов языка Си вынуждали пользователей объявлять все переменные в верхней части функции. Сейчас это уже неактуально. Современные компиляторы не требуют, чтобы все переменные обязательно были объявлены в самом верху функции. В языке C++ приоритетным является объявление (а также инициализация) переменных как можно ближе к их первому использованию.

Такой стиль написания кода имеет несколько преимуществ.

Во-первых, возле переменных, которые объявлены как можно ближе к их первому использованию, находится другой код, который способствует лучшему представлению и пониманию происходящего. Если бы переменная *x* была объявлена в начале функции main(), то мы бы не имели ни малейшего представления, для чего она используется. Для понимания смысла переменной пришлось бы целиком просматривать всю функцию. Объявление переменной x возле операций ввода/вывода данных позволяет нам понять, что эта переменная используется для ввода/вывода данных.

Во-вторых, объявление переменных только там, где они необходимы, сообщает нам о том, что эти переменные не влияют на код, расположенный выше, что делает программу проще для понимания.

И, наконец, в-третьих, уменьшается вероятность случайного создания неинициализированных переменных.

В большинстве случаев, вы можете объявить переменную непосредственно в строке перед её первым использованием.

Правило. Объявляйте переменные как можно ближе к их первому использованию.

Инициализация по умолчанию

Если переменную не инициализировать, то происходит ее инициализация по умолчанию. И переменная получает некоторое значение по умолчанию, которое зависит от места, где эта переменная определена.

Если переменная стандартного типа определена вне функции main, но не инициализирована, то она получает то значение по умолчанию, которое соответствует ее типу. Для числовых типов это число 0.

Например:

```
#include <iostream>
int x;
int main()
{
    int y;
    std::cout <<"X = " << x << "\n";
    std::cout <<"Y = " << y;
        return 0;
}
```

Переменная *x* определена вне функции, и поэтому она получит значение по умолчанию - число 0.

Изменение значения переменной

Ключевой особенностью переменных является то, что мы можем изменять их значения:

```
#include <iostream>

int main()
{
    int x = 6;
    x = 8;
    x = 10;
    std::cout <<"X = " << x; // X = 10

    return 0;
```

Задание типов переменных

Типы переменным в C++ задаются только один раз, изменять их в ходе работы программы нельзя, то есть нельзя изменять размеры выделенных блоков памяти, можно только уничтожать один блок ради создания другого (это возможно при работе с указателями, к которым мы обратимся позже).

В C++ типы делятся на фундаментальные (они же и базовые или примитивные) и составные.

К фундаментальным типам относятся:

- символьные виды типов: char, unsigned char, signed char — относится к целочисленным типам;
- целочисленные виды типов: short int, int, long int, long long int — относятся к целочисленным типам;
- булевский вид типов: bool —относится к целочисленным типам;
- виды типов с плавающей точкой (дробные числа): float, double, long double — относятся к типам с плавающей точкой;
- вид пустого типа: void — относится к неполным типам.

К составным типам относятся:

- массивы,
- функции,
- указатели,
- ссылки,
- классы,
- объединения,
- перечисления.

Последние три типа – из области объектно-ориентированного программирования.

Для работы с числами используют целочисленные типы, эти типы могут быть знаковыми и беззнаковыми. Знаковые типы — это типы для переменных, способных принимать отрицательные значения, а беззнаковые, соответственно, отрицательных значений принимать не способны.

Поскольку числа могут быть положительными и отрицательными, то численным типам необходим знак «плюс» или «минус». Иногда не нужны отрицательные числа, тогда имеет смысл использовать беззнаковый тип, благодаря которому возможно использовать дополнительные положительные числа.

Беззнаковые типы: short int, int, long int - указываются перед переменной, хранящей целые числа. В short int и long int слово int можно опускать: short, long.

Когда нужно использовать свойство знаковости чисел (только положительные, либо не только положительные, но обязательно и отрицательные), используют ключевые слова signed (знаковое) и unsigned (беззнаковое):

```
unsigned x = -1; //x - беззнаковая переменная, поэтому отрицательное
// значение внутри неё быть не может
signed y = -1; //y - знаковая переменная, поэтому внутри неё законно
 //может находиться отрицательное значение
//Выведите на экран значения x и y самостоятельно
unsigned short z1 = 1;
unsigned int z2 = 1; //==>unsigned z1=1
unsigned long z3 = 1;
```

Имейте в виду, что несмотря на то, что при использовании беззнаковых типов появляется возможность использовать дополнительные положительные числа, вместимость самого типа никак не изменяется. Из целочисленных числовых типов для работы программ в большинстве случаев хватает типа int.

Размеры целочисленных типов во всех компиляторах C++ подчиняются следующему неравенству:

```
char <= short <= int <= long <= long long
```

Задание типов переменных

Мир не ограничивается только целыми числами, поэтому существуют типы, называемые типами с плавающей точкой. В отличие от целочисленных типов они не могут быть объявлены как беззнаковые или знаковые:

```
 int main()
{
      unsigned float x = 1; //ошибка компиляции, числа с плавающей
      //точкой не могут быть беззнаковыми
      signed float y = 1; //ошибка компиляции, числа с плавающей точкой
      // не могут быть объявлены знаковыми
      float z = 1; //это работает
      unsigned double a = 1; //ошибка компиляции, числа с плавающей
       // точкой не могут быть беззнаковыми
   }
```

Размер выделяемой памяти для типов с плавающей точкой определяется реализацией компилятора. У чисел с плавающей точкой после точки какое-то определённое число значащих цифр, за значащими цифрами идут цифры, которые представляют собой информационный мусор, поэтому толку от них никакого нет.

Задание типов переменных

В совокупности целочисленные типы и типы с плавающей точкой называются арифметическими типами.

Тип переменной определяет, что именно в ней будет храниться. Хранимым значением может быть число, строка, какой-то наш собственный тип данных. Все типы, даже одинаковые, но с разным названием, воспринимаются компиляторами C++ как типы с разной структурой, то есть считаются разными, поэтому у разных типов переменных могут различаться операции, которые с ними можно использовать.

Так, например, если переменная имеет целочисленный тип, то для нее определена операция сложения (+). Если переменная имеет тип «массив символов», то операции сложения (+) в таком типе нет.

Каждый тип может имеет свои особенности, поэтому при создании переменных вы должны правильно выбирать соответствующий им тип.

Задание типов переменных

Пример кода объявления переменных.

```
int main ()
{
  short x1;         //Знаковый. Короткий диапазон целых чисел
  signed short x2;  //Знаковый. Короткий диапазон целых чисел
  unsigned short x3;//Беззнаковый. Короткий диапазон целых чисел
  int y1;         //Знаковый Диапазон целых чисел
  signed int y2; //Знаковый Диапазон целых чисел
  unsigned int y3; //Безнаковый Диапазон целых чисел
   ong z1;          //Знаковый длинный дипазон целых чисел
  signed long z2;   //Знаковый длинный дипазон целых чисел
  unsigned long z3; //Безнаковый длинный дипазон целых чисел
  char ch1; //или знаковый, или беззнаковый диапазон для кодов
  // символов. Может быть в обоих вариантах.
  signed char ch2; //Знаковый диапазон для кодов символов
  unsigned char ch3; //Беззнаковый диапазон для кодов символов
  //Иногда в кодах символов применяют отрицательные значения, а
  // иногда только положительные.
  // short char; // неверно. char не может быть коротким или длинным.
  // signed double d; //неверно. Знаковыми и беззнаковыми могут быть
  // только целочисленные типы
}
```

В программах используются переменные. Имя переменной выбирает составитель программы; имя переменной должно начинаться с буквы латинского алфавита и может содержать буквы латинского алфавита, цифры и символы подчеркивания. Заглавные и строчные буквы считаются разными.

КОНСТАНТЫ В С++

Каждый элемент данных, используемый в программе, является либо константой, либо переменной.

Константой называется элемент данных, значение которого в процессе выполнения программы не меняется.

В языке С++ используется константы следующих видов:

- числовые,
- логические (булевские),
- символьные,
- строковые.

Числовые константы предназначены для представления числовых данных (целых и вещественных).

Булевские константы используются для представления данных, имеющих смысл логических высказываний (типа: да-нет, истина-ложь, 1-0).

Символьные и строковые константы - это отдельные символы и их последовательности.

ВВОД, ВЫВОД В C++. Библиотека iostream

C++ рассматривает ввод и вывод как поток байтов. (В зависимости от реализации и платформы это могут быть 8-, 16- или 32-битные байты, но все равно они будут байтами.) Однако многие виды данных в программе организованы в виде более крупных блоков, нежели отдельный байт. Например, тип int может быть представлен 16- или 32-битным двоичным значением, а значение типа double — 64- битными двоичными данными. Но при отправке потока байтов на экран желательно, чтобы каждый байт представлял значение символа. То есть для отображения на экране числа -2.34 понадобится отправить пять символов: «-», «2», «.», «3» и «4», а не внутреннее 64-битное с плавающей точкой представление этого значения.

Частью стандартной библиотеки C++ является библиотека iostream – объектно-ориентированная иерархия классов, где используется и множественное, и виртуальное наследование. В ней реализована поддержка для файлового ввода/вывода данных встроенных типов. Кроме того, разработчики классов могут расширять эту библиотеку для чтения и записи новых типов данных.

Библиотека iostream

Для использования библиотеки iostream в программе необходимо включить заголовочный файл:

```
#include <iostream>
```

Операции ввода/вывода выполняются с помощью классов istream (потоковый ввод) и ostream (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод.

Для удобства в библиотеке определены два стандартных объекта-потока:

cin – объект класса istream, соответствующий стандартному вводу; в общем случае он позволяет читать данные с терминала пользователя;

cout – объект класса ostream, соответствующий стандартному выводу; в общем случае он позволяет выводить данные на терминал пользователя.

Вывод с помощью cout

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево (<<), а ввод – с помощью оператора сдвига вправо (>>) – это условные названия:

```
#include <iostream>
#include <string>
int main()
{
 // вывести сообщение на экран
cout << "Здесь появится ваше имя:";
cout << "Вася" <<endl;
}
```

Перегрузка операторов в программировании — один из способов реализации полиморфизма, заключающийся в возможности одновременного существования в одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются.

Вывод с помощью cout

Назначение операторов легче запомнить, если считать, что каждый «указывает» в сторону перемещения данных.

Например,

>> *x*

перемещает данные в *x*,

<< *x*

перемещает данные из *x*.

Поэтому одной из наиболее важных задач, стоящих перед программой, является преобразование числовых типов, таких как int или float, в поток символов, который представляет значения в текстовой форме. Таким образом, транслируется внутреннее представление данных в виде двоичных битовых последовательностей в выходной поток символьных байтов.

Чаще всего мы используем cout с операцией <<, которая также называется операцией вставки:

int clients = 22;

cout << clients;

В языке C++, как и в языке C, по умолчанию операция **<<Выражение** наподобие ***x* <<3** означает: загрузить двоичное представление *x* и сдвинуть все его биты на три позиции влево. Очевидно, что это не слишком тесно связано с выводом. Но операция << определяется для вывода. В этой ипостаси операция << называется операцией вставки, а не операцией сдвига влево. (Операция сдвига влево обретает эту новую роль посредством своего визуального аспекта, который предполагает смещение информации влево.)

Операция вставки перегружена для применения со всеми базовыми типами C++:

- char
- short
- int
- long
- float
- double

Если вы — программист на C и страдаете от многообразия спецификаторов типа % и проблем, связанных с несоответствием спецификатора действительному типу значения, то использование cout покажется вам до неприличия простым. (Разумеется, как и ввод C++ посредством cin.)

Ввод с помощью cin

Теперь обратимся к вводу и передаче данных в программу. Объект cin представляет стандартный ввод в виде потока байтов. Обычно этот поток символов генерируется клавиатурой. При вводе последовательности символов объект cin извлекает эти символы из входного потока. Этот ввод может являться частью строки, значением типа int, типа float или какого-либо иного типа. Таким образом, извлечение символов из потока предполагает также преобразование типа. Объект cin на основании типа переменной, предназначенной для приема значения, должен применять свои методы для преобразования последовательности символов в значения соответствующего типа. Обычно cin используют следующим образом:

cin >> *x*;

Здесь x идентифицирует область памяти, в которую помещается ввод. Это может быть в самом простом случае именем переменной. То, как cin интерпретирует ввод, зависит от типа данных x.

Ввод с помощью cin

Класс istream, определенный в заголовочном файле iostream, перегружает операцию извлечения >> для распознавания следующих базовых типов:

- char
- short
- int
- long
- float
- double

Например, предположим, что имеется следующий код:

```
// Вывод приглашения на ввод имени cout « "Enter your first name:\n";
string name;
// Ввод имени cin » name;
```

Таким образом, можно ввести данные, то есть в этом примере имя.

Программа на языке С++ имеет определенную **структуру**.

Существует последовательность заранее определенных строк кода, которая приведена в табл. 8.

Таблица 8. Часто встречающиеся строки кода программ С++

#include "stdafx.h"	подключение заголовочного файла для сборки проекта. Обязательный пункт в Visual Studio, в других средах не используется
#include <название_библиотеки>	подключение библиотек. Необязательный пункт. Подробно о библиотеках смотреть ниже.
using namespace std;	использование пространства имен.
int main(){} или void main(){}	главная функция программы. Именно она начинает выполняться, когда запускается программа. Обязательный пункт.
Тело_функции_main	в теле функции main записываются действия и операции, предусмотренные алгоритмом. Обязательный пункт.
return 0;} или }	конец программы. Самый последний оператор. Обязательный пункт

1. <iostream> для VisualStudio – это библиотека для работы с консолью (экраном).

cout – оператор вывода данных на экран.

Пример использования:

cout<<"fraza"; //выведет на экран слово fraza; может вывести любой текст.

cout<<x; //выведет на экран число, хранящееся в переменной x.

cin – оператор считывания с клавиатуры.

Когда у пользователя запрашивают число, программа ждет, пока пользователь не напечатает число и нажмет ENTER. Тогда оператор cin записывает это значение в переменную x.

Пример использования:

cin>>x; //присваивает переменной x значение, введенное с клавиатуры.

cin>>x>>y; //присваивает переменной x первое введенное с клавиатуры значение, переменной y – второе.

endl – оператор перевода каретки на экране на следующую строку и возврата курсора в ее начало. Самостоятельно не используется.

Пример использования:

cout<<endl; //курсор перейдет на новую строку.

cout<<x<<endl;//сначала на экране появится число, хранящееся в переменной, потом курсор перейдет на новую строку. Выводимые далее данные будут печататься с новой строки.

cout<<endl<<"fraza"; // курсор перейдет на новую строку, и на новой строке появится надпись fraza.

precision(n) – функция для отображения на экране дробных чисел с n цифрами после запятой.

Пример использования:

cout.precision(3)<<7.897426; //число 7.897426 выведется на экран в виде 7.897.

2. <math.h> - библиотека математических функций. Основные математические функции представлены в табл. 9

Таблица 9. Основные математические функции C++

Математическая функция	Программная запись	Описание
$\|x\|$	abs(x)	Модуль числа.
sin x	sin(x)	Синус числа, аргумент в радианах.
cos x	cos(x)	Косинус числа, аргумент в радианах
tg x	tan(x)	Тангенс числа, аргумент в радианах.
e^x	exp(x)	Экспонента числа.
ln x	log(x)	Натуральный логарифм числа.
lg x	log10(x)	Десятичный логарифм числа
x^y	pow(x, y)	x в степени y.
$\sqrt{x}$	sqrt(x)	Квадратный корень из числа.
arcsin x	asin(x)	Арксинус числа, в радианах.
arcos x	acos(x)	Арккосинус числа, в радианах.
arctg x	atan(x)	Арктангенс числа, в радианах.
π	M_PI	Число π =3.141593

3. **<iomanip>** для VisualStudio реализует инструменты для работы с форматированием вывода. В этой библиотеке есть функция setw(n), которая применяется для вывода на экран значения, под которое отводится n ячеек. Используется при построении ровной таблицы значений функции.

Пример использования:

```
cout<<setw(5)<<x<<setw(5)<<y<<endl;
```

На экран выведутся два числа: первое (1.5) в первых пяти ячейках, второе (-73) во вторых пяти ячейках: _ _ 1 . 5 _ _ - 7 3 .

Часто встречающиеся строки кода программ С++

Чтобы использовать в программе переменную, необходимо совершить 2 действия.

1) Объявить переменную в начале программы, явно указав тип данных для переменной.

Пример:

double x; //вещественная переменная.

int m; //целочисленная переменная.

Если переменная не будет объявлена, но будет использоваться далее в программе, то программа не запустится, компилятор выдаст ошибку.

2) Проинициализировать переменную, т.е. задать переменной начальное значение.

Пример:

```
x = 7.81;
m = 4; z = x + m;
```

Если переменная не будет проинициализирована, то компилятор не выдаст ошибки, но расчеты будут выполнены неверно.

3) Использовать эту переменную далее в программе в расчетах или при выводе на экран.

Для числовых переменных определены простейшие арифметические операции, которые приведены в таблице 10. Для их использования не нужно подключать библиотеку.

Часто встречающиеся строки кода программ С++

Таблица 10. Логические операции С++

Сравнение в С++	Описание	Пример в программе
>	больше чем	x>0
<	меньше чем	y<z
>=	больше или равно	y>=x
<=	меньше или равно	z<=8.56
==	проверка на равенство	x==0.7
!=	не равно	x!=y
&&	логическое И	x>0 && x<1 // двойное неравенство 0<x<1
||	логическое ИЛИ	s<8 || s>10

Накопленный опыт программирования привел к формированию следующих рекомендаций по составлению наглядных и легко читаемых программ.

1. Стандартизация стиля программирования заключается в том, что необходимо всегда придерживаться одного способа программирования, записи программы.

С целью рационального размещения текста, не следует операторы программы писать сплошным текстом.

2. Для четкого выявления вложенности управляющих структур требуется особым образом располагать операторы в тексте, так чтобы служебные слова, которыми начинается или заканчивается тот или иной оператор, записывались на одной вертикали, а все вложенные в него операторы записывались с некоторым отступом вправо. При записи конструкции языка более глубоких уровней вложенности следует сдвигать их от начала строки вправо. Каждое описание и каждый оператор следует писать с новый строки. Перенося описания операторов на новые строки, записи надо сдвигать вправо. Следует по возможности избегать длинных строк.

3. Рекомендуется любую программу сопровождать комментариями, поясняющими назначение всей программы и отдельных ее блоков и функций.

4. Имена для объектов программы надо выбирать так, чтобы они наилучшим образом соответствовали этим объектам, отражали их назначение.

5. Списки идентификаторов в блоках описания следует упорядочивать: это облегчает поиск в них нужных элементов.

6. Программирование ведется сверху вниз. В процессе разработки алгоритма и программы следует начинать с самой общей модели решения, постепенно уточняя ее до уровня отдельного блока, а затем детально прорабатывая отдельный блок.

Особое значение для программиста имеет предупреждение и исправление ошибок в алгоритме и программе решения задачи. Прежде чем выполнит программу, ее текст необходимо ввести в компьютер. Для ввода и изменения (редактирования) текстов используется специальная программа - текстовый редактор.

Так как текст записан на языке программы и непонятен компьютеру, то требуется перевести ее на машинный язык. Такой перевод программы с языка программирования на язык машинных кодов называется трансляцией и выполняется специальной программой-транслятором.

Существует три вида трансляторов: компиляторы, ассемблеры, интерпретаторы.

Интерпретатором называется транслятор, производящий пооператорную (покомандную) обработку и выполнение исходной программы.

Компиляторы преобразуют (транслируют) всю программу в модуль на машинном языке, после чего программа записывается в память компьютера и лишь потом исполняется. Компиляторы анализируют программу и определяют, содержит ли она ошибки. В случае их обнаружения вся работа останавливается. Если же правила языка не нарушены, то формируется модуль на машинном языке, который затем и исполняется.

Ассемблеры переводят программу, записанную на языке ассемблера, (автокоды) в программу на машинном языке.

ТРАНСЛЯТОРЫ. СИНТАКСИС И СЕМАНТИКА

Любой транслятор решает следующие основные задачи:

1) анализирует транслируемую программу, в частности, определяет, содержит ли она синтаксические ошибки;

2) генерирует выходную программу (ее часто называют объектной или рабочей) на языке команд ЭВМ (в некоторых случаях транслятор генерирует выходную программу на промежуточном языке, например, на языке ассемблера);

3) распределяет память для выходной программы (в простейшем случае это заключается в назначении каждому фрагменту программы, всем переменным, константам, массивам данных и другим объектам своих адресов участков памяти).

В отличия от естественных языков, таких как русский, английский и др., язык программирования имеет очень ограниченное количество слов, понятных компиляторам, и строгие правила записи команд. Совокупность этих требований образует синтаксис языка программирования, а смысл команд других конструкций языка - его семантику.

ИНСТРУКЦИИ. ОПЕРАТОРЫ

Алгоритм решения любой задачи состоит из отдельных довольно мелких шагов. В программе для каждого шага алгоритма, в том числе для организации ветвлений и циклов, записывается отдельная инструкции (команда). Таким образом, программа состоит из отдельных инструкций или команд. Эти инструкции в программировании принято называть операторами. Часто в литературе по программированию программу определяют, как последовательность операторов.

Операторы могут объединяться в более крупные конструкции - составные операторы, подпрограммы-функции. Такие конструкции состоят из нескольких элементарных операторов, однако в программе могут использоваться как один оператор.

Подпрограммы-функции универсального назначения могут располагаться в особом образом оформленных файлах - библиотечных модулях.

ПРИМЕР ПРОСТОЙ ПРОГРАММЫ

Сейчас мы напишем простую программу-калькулятор, которая будет принимать от пользователя два целых числа, а затем определять их сумму:

```
#include <iostream>
using namespace std;

int main()
{
   setlocale(0, "");
   /*7*/ int a, b; // объявление двух переменных a и b целого типа данных.
   cout << "Введите первое число: ";
   /*9*/ cin >> a; // пользователь присваивает переменной a какое-либо
   //значение.
   cout << "Введите второе число: ";
   cin >> b;
   /*12*/ int c = a + b; // новой переменной c присваиваем значение суммы
   //введенных пользователем данных.
   cout <<"Сумма чисел = " <<c<<endl; // вывод ответа.
   return 0;
}
```

ПРИМЕР ПРОСТОЙ ПРОГРАММЫ

Разбор кода

В 7-й строке кода программы мы объявляем переменные «*a*» и «*b*» целого типа int. В следующей строке кода выводится сообщение пользователю, чтобы он ввел с клавиатуры первое число.

В 9-й строке стоит конструкция cin >>. С помощью нее у пользователя запрашивается ввод значения переменной «*a*» с клавиатуры. Аналогичным образом задается значение переменной «*b*».

В 12-й строке мы производим инициализацию переменной «*c*» суммой переменных «*a*» и «*b*». Далее организуется вывод с помощью cout, который выводит на экран строку и значение переменной «*c*».

При выводе переменных мы не заключаем их в кавычки, в отличие от строк.

ПРИМЕР ПРОСТОЙ ПРОГРАММЫ

Задание для самоконтроля.

Попробуйте провести несколько экспериментов с программой — сделайте аналогичный пример с умножением или вычитанием переменных. Не бойтесь издеваться над программным кодом, потому что ошибки — неотъемлемая часть обучения любому делу. И не забываем про точки с запятой.

Литература

1. Процедурное программирование на языках СИ и С++ : учебно-методическое пособие / Л. А. Скворцова [и др.]. — М.: РТУ МИРЭА, 2018. — 238 с. [Электронный ресурс]. Режим доступа: https://library.mirea.ru/books/53585
2. Трофимов В.В., Павловская Т.А. Алгоритмизация и программирование: учебник для академического бакалавриата. М.: Издательство Юрайт, 2017. [Электронный ресурс]. Режим доступа: https://www.intuit.ru/studies/courses/16740/1301/info
3. Уроки С++ с нуля. [Электронный ресурс]. Режим доступа: https://code-live.ru/tag/cpp-manual
4. Введение в языки программирования Си С++. [Электронный ресурс]. Режим доступа: https://www.intuit.ru/studies/courses/1039/231/info